UNIVERSITY OF ILLINOIS
DIGITAL COMPUTER LABORATORY
GRADUATE COLLEGE

Report No. 132

A PROGRAMMING LANGUAGE FOR THE PARALLEL PROCESSING OF PICTURES

by

R. Narasimhan

January 9, 1963

This work was supported in part by the
Atomic Energy Commission Contract No. AT(11-1)-1018

1. INTRODUCTION

1.1 The Background.

In a recent report* a particular descriptive scheme was suggested for the analysis of pictures composed of line-like elements. Briefly, the scheme envisaged a two-stage processing technique. In the first phase, the input picture was converted to a labeled graph--with labeled vertices and branches--by means of certain well-defined labeling algorithms. In the second phase, the labeled graph so obtained was reduced to a network composed of certain primitive strings. This reduction was sought to be accomplished by making use of given "grammar rules" for string manipulation. These grammar rules were to be defined in terms of the labels generated in the first phase and contained in themselves an implicit characterization of the class of pictures under consideration.

While, ultimately, the adequacy of such a model as this has to be based on extensive empirical verifications, it might be of some interest to consider the following plausibility argument: the correspondence between a picture composed of line-like elements and a graph is intuitively evident. That, in fact, a graph with labeled vertices and branches can be constructed needs substantiation and this is verified by the explicitly given algorithms. The connection matrix of this graph which describes it is clearly also a description of the underlying picture. By making use of the vertex labels the original matrix can be replaced by a reduced version incorporating only a subset of vertices. The matrix entries will now be strings (of labeled branches) rather than single branches. Grammar rules can now be applied to transform and compose these strings into certain primitive categories (or types). Finally, the matrix can be rewritten to represent a network made up of these primitive strings. For a given class of pictures, the total descriptions scheme, then, is built out of the following:

1. An alphabet of primitive strings, and

2. A set of "well-formed" networks of these primitive strings.

---

* R. Narasimhan: A Linguistic Approach to Pattern Recognition, Report 121, Digital Computer Laboratory, University of Illinois, (1962).

CONTENTS

Any given input picture is now to be described in terms of a statement that it consists of such and such primitive strings put together in such and such a network. This rather close analogy to language models and to the hierarchic levels in linguistic analysis is more than incidental. This was the reason for referring to this model in the earlier report as a linguistic approach to pattern recognition.*

1.2. Present Report.

To argue the adequacy of this model, the earlier report contained several examples of input pictures processed according to the details set forth in the scheme, making use of a few very rudimentary grammar rules. The principal concern of that report, however, was one of exposition of the model and a discussion of its feasibility and some of its intrinsic merits for the treatment of the pattern recognition problem. Formalization of the processing details and their machine realization were postponed for later consideration. We return to this aspect of the problem now and develop in this report a programming language in terms of which the labeling algorithms of phase I of our model can be effectively described and studied.

Apart from the efficient description of the algorithms, a primary motivation for the formulation of the programming language is to aid in the system design of a computer in which these algorithms can be mechanically realized most readily; for, as was argued in the earlier report, it is our contention that the preprocessing of pictures (i.e., noise cleaning and other standardization aspects) and the labeling algorithms are intrinsically more efficiently handled in a parallel processing computer. One method of characterizing the structure of such a computer (for picture processing) would be to try and define, purely formally, the types of functions it should be capable of computing i.e., what types of functions on what types of structured

---

* In "A System for the Automatic Recognition of Patterns" (Proc. I.E.E. 106(B), (1959), 210; see also, Comp. Jour. 4 (1961), 129), Grimsdale et al describe a similar "linguistic" model. This work was brought to our attention subsequent to the publication of Report 121 and so was not referred to there. It is worth noting that Grimsdale et al restrict their consideration to the recognition of alphanumeric characters and their primary concern is the efficient realization of their processing algorithms within the context of conventional arithmetic computers.

operands yielding what types of values. A good systems organization for the computer can then be considered as one which mirrors the structured operands in a natural way and has wired-in primitive operations in terms of which the required functions can be computed under programmed control. As applied to picture processing we shall consider this question in some detail in the next section. As we shall see there, by this procedure we arrive at a system structure which is almost identical to that currently envisaged for the PAU on the basis of quite different considerations.*

This, however, only solves partially the task we started out with. To complete the formalization of the parallel processing algorithms, it is necessary to develop an algorithmic language in terms of which a computer such as the one envisaged in the previous paragraph can be formally programmed. A natural approach to this problem would be to consider whether our structured operands and the special functions defined on them could be embedded in a suitable way in an extension of an already well-developed programming language. We shall see that in fact this is readily accomplished and make use of a suitable extension of ALGOL-60 to this purpose. These details will be considered in section 3 where we shall also give some examples of parallel processing routines written in this language.

From a detailed consideration of picture processing routines we have now reduced parallel processing to computations of certain classes of functions defined on certain types of structured operands. It is natural to wonder at this stage whether the programming language so developed has any applicability to problems other than those which arise in picture processing. An answer to this question would also help to define a general purpose parallel processing computer applicable to a wider variety of problems. In section 4 we shall provide a partial answer to this by showing that address computations for content-oriented memories (the so-called associative memories) are strictly computations of functions on labels (or tags) included in the class of functions we have been considering and hence can be handled efficiently within our formalism. It is also our feeling that, when properly formulated,

---

* B. H. McCormick: Design of a Pattern Recognition Computer. Part II: The Pattern Articulation Unit (PAU); Report 122, Digital Computer Laboratory, University of Illinois; (1962).

one aspect of information retrieval problems should be expressible in terms of computations of suitably defined functions on labels (or indexes) whose values are documents. A parallel processing computer with a basic structure similar to that described in this report would seem to be applicable for the computation of a large class of functions of this type.

2. A FORMALISM FOR PICTURE PROCESSING

2.1. Introduction.

In this section we shall first develop a formalism to describe parallel processing and then introduce a variety of special functions of immediate use in picture processing. The motivations for the several details should be fairly obvious intuitively or from the context. However, since the entire approach takes as its point of departure the specific processing schema discussed at length in DCL Report 121 (see footnote in Sec. 1), a knowledge of its contents might add to the perspicuity of the treatment which follows.

2.2. The Formalism of Computation.

The basic domain of operation for parallel processing is a fixed, finite array of points. Without loss of generality we shall assume this to be a square array of points.* Let P be a generic symbol denoting the points of this array. P can assume values 0 or 1. A particular assignment of values to the points of the array constitutes a picture. Typically, in parallel processing, we start with an initial picture and generate from it, by well-defined operations, other pictures. In practice this is done by modifying the values of some of the P's in the old picture. It is important to note that, in general, we do not modify the values of all the points every time. Central to parallel processing, then, is the computation of functions defined over a square array of points, the values of the functions being dependent on specified pictures. In other words, the computation is done with respect to a specified picture. So, to specify a particular step in parallel processing, we have to specify a function, a picture and a set of points where the values of the function are to be computed with respect to this picture. However, within the parallel processing schema, we cannot operate on individual points or even on sets of points by only on the entire array of points. The only way we can refer to a particular set of points is by constructing a characteristic picture of that set, i.e., a picture in which the points of the set have

* For example, in the PAU under design the square array has a bit size of 40x40 with a central picture area of 32x32.

the value 1 and all other points the value 0. Purely formally, then, we can say that each step in parallel processing consists in computing a given function with respect to two pictures--the <u>argument</u> and the <u>context</u>, respectively--and generating a third picture.

To carry out such a computation we need at least 3 replicas of the basic square array. We will obviously need many more replicas to perform a sequence of such computations. Let us, then, visualize an entire column of such arrays, stacked one on top of another and assume that computations are performed within this set-up. This already provides a preliminary specification for a system structure to realize these computations. But we shall postpone till the end of this section further consideration of the details of this problem.

Let S be a generic symbol representing these arrays and let particular pictures (i.e., specific arrays to whose points values have been assigned) be distinguished by the addition of subscripts.* A step in parallel processing can now be formally represented by a statement of the form

$$S_k := F(S_i, S_j)$$

with the following convention: l.h.s. is the resulting picture; the first symbol in the function (on r.h.s.) refers to the argument set (i.e., its characteristic picture) and the second symbol to the context. Sometimes, in defining a picture, the context may be left unspecified. In all such cases it is understood that the context is the <u>universe</u>, i.e., the picture with all P values equal to 1. As we shall see presently, the computations will, in general, depend not only on the context picture but also on certain given parameters. Thus the general form of parallel processing computation should actually be represented as follows:

$$S_k := F(S_i, S_j; \text{ parameter}).$$

2.3. <u>Some Special Functions for Picture Processing</u>.

Our next task is to define a set of functions in terms of which useful parallel processing routines can be developed. These functions will

* It must be emphasized that this is just for notational convenience. These quantities should not be confused with subscripted variables in the sense of ALGOL-60. This is important to remember especially when we write programs in ALGOL-60 later on in this report.

clearly form a hierarchy in the sense that some of them will be primitive, others defined in terms of these (i.e., as routines involving these), and still others in terms of this new set and so on. In designing a computer to realize these functions one would naturally want the machine orders to fit into this hierarchy at a level such as to provide a good match between the programming language and the machine language. We shall not, however, consider this problem in its complete generality here. In an appendix to this section we list a preliminary set of functions which we have arrived at on the basis of our work in picture processing. All the algorithms that have been developed so far can be efficiently described in terms of one or more of these functions*. For our persent discussion we restrict ourselves to a consideration of these functions and introduce some additional notations and definitions towards this end. (As far as possible we use the terminology introduced in Report 121):

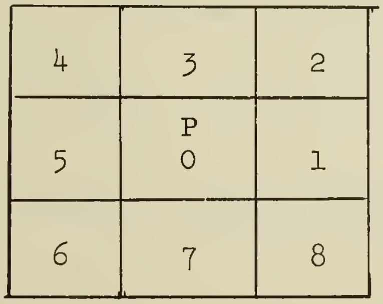

Fig. 1

<u>Mark and Chain</u>. For any point P, we shall denote by n(P) the set of 9 points consisting of P and its eight immediate neighbors in the array. The individual points of this set will be denoted by $N_i(P)$, i = 0, 1, 2, 3, ..., 8, where the subscript convention is as shown in Fig. 1. By a <u>direction list</u> (or <u>direction</u> or <u>dir</u>, for short) we shall mean an index string $i_1 i_2 \ldots i_k$ with $k \leq 9$ and each $i_j = 0, 1, \ldots,$ or 8, no index value being repeated in the string. Using a direction list we can refer to a subset of N(P) as follows:

---

* Several of these algorithms were developed in conjunction with B. H. Mayoh and R. K. Rice and will be reported on separately.

A parallel processing simulator for IBM-7090 to realize this set of functions has been written by James H. Stein and will be described by him in a separate report.

$$\sum_{i\in \text{dir}} N_i(P) \equiv \left\{Q \;\middle|\; Q = N_i(P) \text{ for some } i \text{ in the direction list}\right\}$$

If S is any picture, we can define a context--dependent subset of N(P) as follows:

$$N(P,S) \equiv \left\{Q \;\middle|\; Q\in N(P) \text{ and } Q\in S\right\} .$$

Clearly we can extend this notion of context-dependence to subsets of N(P) defined by a direction list.

In terms of these sets, the first two neighborhood operations listed in the appendix can be described.

MARK (S; direction)

$$= \left\{Q \;\middle|\; Q\in \sum_{i\in \text{dir}} N_i(P) \text{ and } P\in S\right\} .$$

CMARK (S_1,S_2; direction)

$$= \left\{Q \;\middle|\; Q \in \text{MARK }(S_1;\text{ direction}) \underline{\text{ and }} Q \in S_2\right\} .$$

Thus, the operation MARK marks all those immediate neighbors of the points in S specified in the direction list. CMARK is a context-dependent operation which selects a subset of the above marked set which is also included in the context picture.

Given a picture S and a point P in it, an <u>i-chain</u>, ($i = 1,2,\ldots,8$), through P is the longest line of points $P_{k_1} P_{k_2} \ldots P_{k_m}$ such that $P_{k_j} \in S$ for $j = 1,2,\ldots,m$ and $P_{k_1} = P$ and $P_{k_j} = N_i(P_{k_{j-1}})$ for $2 \leq j \leq m$. (For further details and illustrations, see Report 121.) Clearly, an i-chain through P is a context dependent set. We shall denote it by $\omega_i(P,S)$. The operation CHAIN may now be described as follows:

CHAIN (S_1,S_2; direction)

$$= \left\{Q \;\middle|\; Q \in \omega_i(P,S_2) \text{ for some } i \in \text{dir} \underline{\text{ and }} \text{some } P \in S_1\right\} ,$$

THRESHOLD: Denote by Weight N(P,S), the number of points in N(P,S) and by length $\omega_i(P,S)$, the length of the chain $\omega_i(P,S)$. We can now define the following new sets:

$$T(N(S_1,S_2) \geq m) \equiv \left\{P \mid P \in S_1 \text{ \underline{and} weight } N(P,S_2) \geq m\right\} ;$$

$$T(\omega_i(S_1,S_2) \geq m) \equiv \left\{P \mid P \in S_1 \text{ \underline{and} length } \omega_i(P_, S_2) \geq m\right\} .$$

Here m is a positive integer. It is evident that these definitions can be extended to the general neighborhood operations MARK, CMARK and CHAIN and to the other relational operators, =, <, >, $\leq$. This extension yields us the general THRESHOLD operation as given in the appendix.

CONNECT: Given a picture S and a point P in it we define the k-th iterate of N(P,S), written as $[N(P,S)]^k$, by the following routine, (in writing this routine we anticipate some of our discussions in Sec. 3. of this report) :

Let us generate $[N(P,S)]^k$ in S_1 which is assumed to contain, to begin with, P alone.

```
begin
    for  x := 1 step 1 until k do
        $S_1$ := N($S_1$,S)     end
```

In the above, by $N(S_1,S)$ we have denoted an obvious extension of the set N(P,S) to all points P in S_1. Define now a function

$$C(P,S) \equiv \left\{Q \mid \text{for some } k,\ Q \in [N(P,S)]^k\right\} ,$$

i.e., the set of all points Q such that for some k, Q is contained in the k-th iterate of N(P,S). Clearly, C(P,S) defines the set of all points connected to P in the picture S. It is quite straightforward to extend this connectivity operation to the case where the function N(P,S) is replaced by the more general neighborhood operation CMARK. It is easily verified that CHAIN is actually a particular example of this more general connectivity operation. The operation CONNECT SET given in the appendix is readily composed out of the function C(P,S) defined above.

ORDERING: Superimpose an x-y coordinate system on the square array in the obvious manner. Let $x(P)$, $y(P)$ denote the x,y coordinates of P. Using these we can now order the points of the set S as follows:

for $P_1, P_2 \in S$ we shall write $P_1 < P_2$ ("is smaller than") provided,

$$x(P_1) < x(P_2)$$

$$\text{or} \quad x(P_1) = x(P_2) \quad \underline{\text{and}} \quad y(P_1) < y(P_2).$$

We can now refer to the first (i.e., smallest) point of S and last (i.e., largest) point of S.

It is clear that many of the concepts of pointset topology can be borrowed for our use. For example, in a quite straightforward manner, we can define the notion of a simply-connected region, a rectangle, a rectangular cover for a set S and so on. It is clearly possible to define parallel processing operations involving these. However, for our immediate purpose these additional functions are not required and hence we shall not discuss their implementation here.

2.4. A System Structure for Machine Realization.

In 2.2 above we have already seen that the most natural machine organization to realize parallel processing computations consists of a stack of two-dimensional registers (referred to as planes) arranged one on top of another in the form of a column, each being a replica of our basic domain of operation, i.e., the square array. In terms of the special functions we have been discussing in 2.3, it is evident that the control unit associated with this stack should be capable of performing certain primitive set-operations and thresholding operations.

Consider now a stack consisting of m planes. Denote by $z(P)$, the m-bit word consisting of the point P in each plane; (see Fig. 2). As has already been mentioned, in picture processing, typically, we start with the original picture in some plane (say, plane S_1) and, by well-defined algorithms, we assign labels to each point P of this picture (i.e., we generate a sequence of pictures S_{i_1}, S_{i_2}, ..., each corresponding to a particular computed label). Thus, at any moment during the processing, the word $z(P)$ contains all the

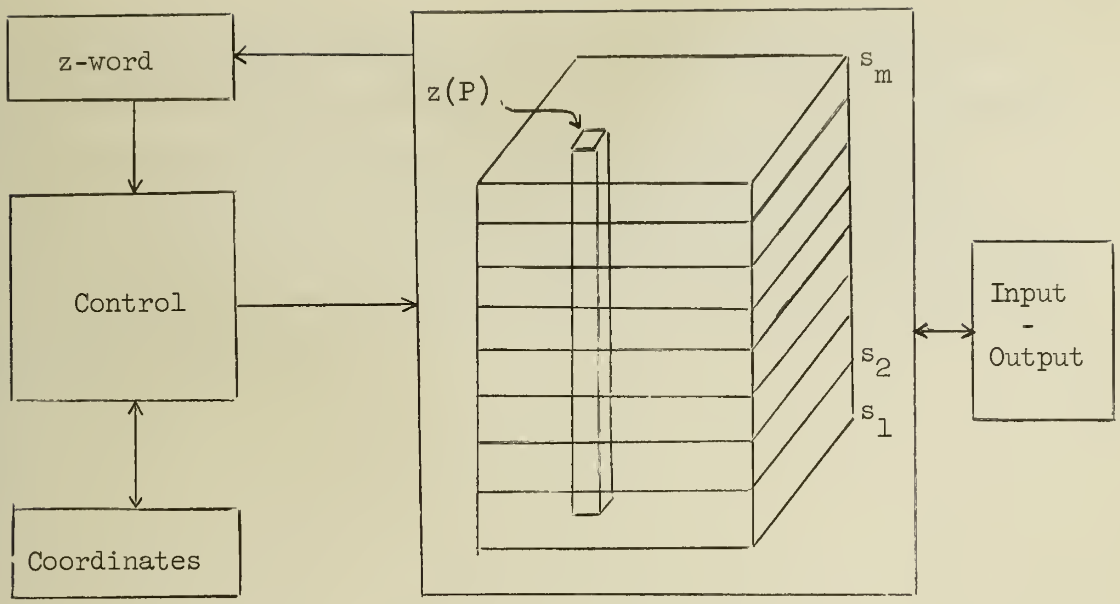

Fig. 2

presently generated labeling information about the point P. The natural thing to do would be to make use of this information in determining the course of the processing sequence. The facility required to be able to perform such conditional branching operations is the ability to read the word z(P) for any specified point P. Closely related to this requirement isthe ability to list the coordinates of a specified set S of points.

Figure 2 shows in a very rudimentary form a system organization for a parallel processing computer. It will be seen that, schematically, this organization is almost identical to that described by McCormick in Report 122 (see footnote in Sec. 1) for the design of the Pattern Articulation Unit. In this actual design, the division of the stack of planes into a "computing" part (the stalactites), and a storage part (the transfer memory from which the z-words are readout), has been motivated solely by hardware considerations and by the resulting simplification in circuit realization. From his treatment it will also be evident that the order code and the micro-operations discussed at length in his report form a complete set of primitives to realize all the special functions we have been considering. All this, of course, is

not totally surprising since both these efforts stem from the same set of preliminary studies in picture processing. But what is of great interest is that a system organization arrived at from micro-considerations (i.e., hardware/circuit oriented) and one arrived at from macro-considerations (i.e., problem oriented) should both turn out to be so nearly identical thus showing that the resulting design is canonical, in some sense, for the class of problems under consideration.

APPENDIX TO SECTION 2

A PRELIMINARY LIST OF FUNCTIONS FOR PICUTRE PROCESSING

(Note: S_1, S_2, S_3 are pictures, not necessarily different. For explanations about the functions, see text. A complete description of all the functions, including those mentioned only by title in this list, will be published separately by J. Stein, along with an account of his IBM-7090 simulator.)

Group A: Set Operations.

1. $S_1 := 0$
2. $S_1 := S_2$
3. $S_1 := \overline{S}_2$
4. $S_1 := S_2 + S_3$
5. $S_1 := S_2 * S_3$
6. $S_1 := S_2 - S_3$ (i.e., $S_2 * \overline{S}_3$)
7. $S_1 := S_2 \not\equiv S_3$ (i.e., $S_2 \oplus S_3$)

Group B: Neighborhood Operations.

8. $S_1 :=$ MARK (S_2; direction)
9. $S_1 :=$ CMARK (S_2, S_3; direction)
10. $S_1 :=$ CHAIN (S_2, S_3; direction)

(Note: 1. direction is to be specified by a direction list of the form $i_1 i_2 \ldots i_k$ where $k \leq 9$ and each $i_j = 0, 1, \ldots, 7$ or 8, without repetitions.

2. In (9) and (10), S_2 is the argument and S_3, the context.)

Group C: Threshold Operations.

11. $S_1 :=$ THRESHOLD (Weight Function; Relational Operator; m).

where: Weight Function ::= MARK | CMARK | CHAIN
Relational Operator ::= $<$ | $\leq$ | $=$ | $\geq$ | $>$
m is a positive integer.

(Note: In a threshold operation, <u>always</u>, $S_1 \subseteq S_2$ where S_2 is the argument in the weight function.)

<u>Group D: Boolean Operation</u>.

12. <u>If</u> $S_1 = 0$ <u>then</u>

13. <u>If</u> $S_1 \neq 0$ <u>then</u>

<u>Group E: Connectivity Operation</u>.

14. CONNECT (P,S_2)

15. CONNECTSET $(P,S_2;S_3)$

(Note: P is a <u>source</u> point; S_2 is a tag or label. S_3 is a <u>sink</u>, usually a set of (connected) sets. In (14), all points in S_2 connected to P are marked. In (15), the members of S_3 connected to P by the label S_2 are listed. (See below for <u>list</u>.))

<u>Group F: List Operation</u>.

16. LIST POINTS (S_1)

17. LISTSETS (S_1)

(Note: In (16), the coordinates of the points in S_1 are listed. In (17), S_1 is a set of (connected) sets and the operation lists one representative point for each set in S_1.)

<u>Group G: z-Read Operation</u>.

18. Read z(P).

<u>Group H: I/O Operation</u>.

19. Input

20. Output .

(Note: These are picture input/output operations.)

3. A PROGRAMMING LANGUAGE

3.1. The Structure of ALGOL-60.

In Section 2, from a detailed consideration of the parallel Processing functions required for operating with pictures, we arrived at a rudimentary structure for a computer to carry out these operations under programmed control. To complete the task of formalization of picture processing algorithms, it is now necessary to describe a formal programming language in which such a computer can be programmed. As we remarked in the introduction, a natural approach would be to examine whether this formal language could be obtained as a suitable extension of some already well-developed programming language. In this section we shall try to accomplish this by using ALGOL-60 as our base language. But before attempting this, let us first recall that our motivation throughout these discussions has been two-fold: (1) to develop a formalism to describe picture processing; (2) to define a computer structure which would be a good match to the structure of our processing algorithms. With regard to (2), we have already reached some definite conclusions on the basis of our considerations in the previous section. Thus, in trying to extend ALGOL-60 to suit our purposes, we should want the extension to fit into this ground that has already been partially prepared.

To do this, we shall first study the structure of ALGOL-60 from the point of view, so to speak, of a system designer. Table I gives such an analysis. The column on the left is an approximate type-classification from the system view point. The middle column lists the syntactic categories of ALGOL-60 and that on the right, some of the implied semantics. The hierarchic levels of the language are immediately evident from the table. An analysis such as this also highlights very clearly some of the inherent inadequacies of the language, as at present constituted, both to match the algorithms it has been set up to specify and to match the structure of the class of computers presently available to realize the algorithms. Since some of the points are of broad, general interest and also have a bearing on our overall approach, we shall digress a little to discuss these.

To begin with, it is clear that this language is designed preeminently to describe the algorithms of arithmetic computations. However, the power and

ALGOL-60: Structure

1. Primitive Symbols:	1. Letters	
	2. Digits	
	3. T-values	
	4. Punctuations	
2. Primitive Operands:	1. Identifier	Simple variable, array, label, switch, procedure
	2. Number	
	3. String	
3. Primitive Operations:	1. Arithmetic	$+ \ - \ \times \ / \ \div \ \uparrow$
	2. Relational	$< \ \leq \ = \ > \ \geq \ \neq$
	3. Logical	$\equiv \ \supset \ \vee \ \wedge \ \neg$
4. Primitive Functions:	1. Arithmetic Expression	Computes a number
	2. Boolean Expression	Computes a T-value
	3. Designational Expression	Computes a label
5. Primitive Subroutines:	1. Assignment Statement	
	2. Goto Statement	
	3. Conditional Statement	
	4. For Statement	
	5. Procedure Statement	
	6. Dummy Statement	
6. Routines:	1. Block	Input Specification
		Declaration:
		1. Type declaration
		2. Array declaration
		3. Switch declaration
		4. Procedure declaration

TABLE I

flexibility of the language in this respect are severely circumscribed by the exclusive restriction of its primitive operations to scalar operations. Thus vector and matrix operations can only be described within the language in terms of the operations on their components. This limitation becomes all the more acute when it is realized that, at the operand level, the language, even as at present constituted, is able to point to and name non-scalar quantities. It might be argued that this is not as serious a limitation as it might seem at first since the language is intended as an input language for computers which themselves are capable of only scalar operations. A little consideration will, however, show that this argument is deficient in several respects. Firstly, this would imply that ALGOL-60 is in fact a machine-oriented language, (viz., to the class of existing arithmetic computers), contrary to the prevailing opinion. But, even granting this, it will be seen that the present version of the language is ill-matched to the existing structure of these machines; for, practically all these machines are capable of performing some vector operations on binary vectors (like, logical sum, logical product, cyclic permutation, etc.) and, as we have just seen, these operations cannot be expressed within the language except in a cumbersome way. These observations are, of course, not very original and most of these limitations we have been discussing are quite well known. But it is our view that the significance of the inadequacies is especially perspicuous when the language structure is viewed simultaneously from the points of view of system design and algorithmic descriptions.

3.2. A Formal Extension of ALGOL-60.

Table I also shows clearly the directions in which extensions both to the language and to a computer might profitably be attempted to improve the matching of their respective internal structures. For a given language structure, it also provides a basis for comparing two different computers as regards the efficiency with which they match that language. However, our main concern here is to consider a particular extension of ALGOL-60 in order to match efficiently descriptions of picture processing routines with a parallel processing computer whose structure we have already outlined.

In Table II we have indicated one such extension to ALGOL-60. The notations in this table correspond to those in Table I and only the additions

Extensions to Picture Processing

2. Primitive Operand:	1. Identifier	Picture, direction
3. Primitive Operations:	4. Picture operations	Sum, product, complement mark, threshold, list, etc.
4. Primitive Functions:	4. Picture Expression	Computes a picture. Boolean expression can be extended to include picture expressions; e.g., If clause::= If (< picture expression> = 0) then
6. Routine:		Input Specifications: 1. Type declaration::= real | integer | Boolean | picture 5. Direction Declaration

TABLE II

have been shown. As might have been expected, we haveincorporated in the language Picture as a new structured operand and have included the primitive operations required to realize the functions we have so far considered. Thus, a simple variable can identify a picture and must be declared to that effect in type-declaration. It is clear that a picture expression can be recursively defined completely analogous to an arithmetic expression. It is also clear that a Boolean expression can similarly be extended to incorporate picture expressions. We shall not consider the formal details here.

Direction is introduced in the language somewhat like a switch. The following formal definitions should make this clear:

```
< direction identifier > ::= < identifier >
< direction number >     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
< direction list >       ::= < direction number > |
                             < direction list > < direction number >
< direction declaration> ::= direction < direction identifier >
```

Values can be assigned to a direction by means of an assignment statement in the usual way:

direction identifier := direction list .

It will be noticed that we have made no special provision to avoid repetitions in the direction list. This, however, is of no great consequence since the semantics can be set up to ignore any such repetitions that might occur. Also, to describe the "components" of a direction individually, it might be of assistance to introduce an additional definition:

<direction component> ::= <direction identifier>[<direction number>].

Clearly, a required semantic restriction is that a direction component exists for a particular direction identifier only if the corresponding direction number is included in the direction list that has been assigned to that identifier.

It is readily verified that this extended ALGOL-60 is quite adequate for the formal specification of all the picture processing functions we have considered so far and the routines based on them.

3.3. Examples of Picture Processing Routines.

Since our principal aim in providing these examples is one of illustration, we shall not attempt to be rigorously formal in the use of the language. We shall make explicit use of the functions listed in the appendix to Section 2 and shall also not trouble to indicate set operators by any special symbols, in as much as their use should be clear from the context.

Example 1: To label α-roads in a given picture S_1. It is assumed that the width of roads ≤ 4. (See Report 121 for explanations by the algorithm. α- is the right-diagonal direction.) The roads are labeled in S_2.

```
begin   S4 := 0 ;
    for K := 1 step 1 until 4 do
begin   S2 := THRESHOLD (CHAIN (S1,S1;48) = K);
        S3 := THRESHOLD (CHAIN (S2,S2;26) ≥ 2 K);
        S4 := S3 + S4   end ;
        S2 := CHAIN (S4,S1;26)   end
```

Example 2: Given a picture S_1, remove P if $N_5(P)$ and $N_7(P)$ are present and $N_2(P)$ is absent. This is one step in a thinning routine that has been used with some success.

```
begin
        $S_2$ := THRESHOLD (CMARK($S_1$,$S_1$; 57) = 2) ;
        $S_3$ := THRESHOLD (CMARK ($S_1$,$S_1$; 2) = 0) ;
        $S_2$ := $S_2$ * $S_3$;
        $S_1$ := ($S_1$ - $S_2$)      end
```

Example 3: Let S_1 be the picture of a road, say a north-south road. We wish to mark the "center-line" of this road in S_2. (The "line" will have a thickness ≤ 2)

```
begin
        $S_2$ := $S_1$;
    1:  $S_3$ := THRESHOLD (CMARK($S_2$,$S_2$;015) = 3);
        $S_4$ := ($S_2$ - $S_3$) ;
        $S_5$ := THRESHOLD (CMARK($S_4$,$S_3$;15) = 1) ;
        If ($S_5$ = 0) then goto 2;
        $S_2$ := ($S_2$ - $S_5$); goto 1;
    2:  end
```

4. APPLICATIONS TO OTHER AREAS

4.1. Address Computation in Associative Memories.

As we pointed out in Section 1, having developed a formalism to represent parallel processing algorithms exclusively within the context of computations on pictures, it is natural to wonder whether the formalism has any applications to problems arising in other areas. In this final section we shall show that problems arising in at least one other area fall within the class of computations we have been discussing. These are problems which require computations of addresses in an associative (i.e., content-oriented) memory. In a recent paper* Falkoff has considered this class of problems at length and has suggested definite algorithms for parallel-search of these memories. His is by far the most comprehensive treatment of the problem that we know of and we shall take that as our point of departure. In what follows we shall show that all the computations performed by his algorithms are strictly within the scope of the parallel-processing schema we have been outlining and hence can be readily handled by the formalism developed for this purpose.

Consider an associative memory of r locations. With each location is associated a tag word of n bits, say. Call the tag word associated with location k, T_k. T_k is a binary vector of dimension n, for $k = 1,2,\ldots,r$. In a simple search, (based on equality), the main problem is the computation of a function of two given binary vectors of dimension n each. These are referred to as the mask ($\underline{m}$) and the argument ($\underline{x}$). The value of the function is an index set S such that, for

$$k = 1,2,\ldots r; \qquad k \in S \Longleftrightarrow i^{th} \text{ bit of } T_k = i^{th} \text{ bit of } x \text{ for each } i \text{ such that } i^{th} \text{ bit of } m = 1.$$

The significance of this search is obvious. It identifies those memory locations whose tagwords coincide with the argument word at all bit positions identified in the mask word. Falkoff specifies several algorithms to compute this function using the notation of Iverson's programming language. Let us

* A. D. Falkoff: Algorithms for Parallel Search Memories; Jour. A.C.M. 9 (1962), 488-511.

first show that within the framework of the parallel processing computer introduced in Sec. 2 (see Fig. 2) this computation is almost trivial to carry out.

We shall arrange the associative memory structure as follows: each point P of the square array stands for a distinct memory location. Thus an rxr array can represent r^2 locations. Let the computer have m planes in the stack, $(m > n)$, and let the first n bits of each z-word, z(P), be the tag word associated with that memory location. Thus each of the planes $S_1, \ldots, S_n$ contains one bit of the tag word of all the r^2 locations. Writing x[i] and m[i] for the i^{th} bits of the argument word and mask word respectively, the following simple routine, clearly, computes the value of the address function of a simple search. Let the result be computed in plane S_m, say:

```
begin     $S_m := 0$ ;
          $S_m := \overline{S}_m$;
      for  i := 1 step 1 until n do
          begin if m[i] = 1 then
begin if x[i] = 1 then  $S_m := S_m * S_i$
                  else  $S_m := S_m * \overline{S}_i$     end
                                          end    end
```

A list operation will now list the coordinates of the memory locations which are the result of the search. This routine corresponds to the serial-by-bit algorithm of Falkoff.

Next, Falkoff considers algorithms for what he refers to as complex searches. For these, the tag words are looked upon as binary numbers and the search is based on comparisons on the magnitudes of these numbers. Specifically he treats search operations based on selecting the locations $\{k\}$ with tag word $T_k \geq T_\ell$ for all $\ell = 1,2,\ldots,r$. Similarly, for $\leq$ replacing $\geq$; the locations whose tagwords lie within specified limits and so on. All these algorithms are based on the positional notation of number representation and involve systematic path computation in the search tree depending on the states of the tag words at each bit position. It is clear from our earlier discussion and the structure of the parallel processing programs that all these address

computations are performed with exceptional ease with a computer set-up such as the one we have been considering. To exhibit this explicitly, we shall consider here the address-computations based on maxima-comparison.

For definiteness, let us suppose that the tag words have their most significant bit position at plane S_1. Let us introduce a Boolean function on pictures called WEIGHT(S) = 1 whose value is True if S contains exactly one point and False otherwise. Clearly, this computation can be readily made using the list operations already introduced. The required program can now be written using this function. We shall again compute the result in S_m.

```
begin     $S_m := 0$ ;
          $S_m := \overline{S}_m$;
      for i  := step 1 until n do
begin
       if  $S_i \neq 0$  then
           begin   $S_m := S_m * S_i$ ;
                   if WEIGHT($S_i$) = 1 then goto 1 end end;
          1: end
```

One major aspect of information retrieval--the so-called "reference-providing" aspect--involves computing complicated Boolean functions on tag words. In concluding this section, let us remark that all such computations are performed with maximum speed and ease within the frame work of a parallel processing schema of the type we have been discussing in this paper.

* * * *